

Vorlesungsmitschrift

# Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b> (Vorlesung 1 am 17.04.)	<b>1</b>
1.1	Was sind Programmiersprachen? . . . . .	1
1.2	Mehrdeutigkeit in natürlichen Sprachen . . . . .	1
1.3	Formalisierungsmethoden . . . . .	1
1.4	Referenzsprache . . . . .	2
1.4.1	Definition der Syntax . . . . .	2
<b>2</b>	<b>Operationelle Semantik</b> (Vorlesung 2 am 24.04.)	<b>4</b>
2.1	Operationelle Semantik am Beispiel der Terme . . . . .	4
2.2	Terme der Sprache WHILE . . . . .	4
2.3	Informelle Semantik . . . . .	4
<b>3</b>	<b>Operationelle Semantik</b> (Vorlesung 3 am 08.05.)	<b>7</b>
3.1	Allgemeines . . . . .	7
3.2	Operationelle Semantik von WHILE . . . . .	7
<b>4</b>	<b>Reduktionssemantik</b> (Vorlesung 4 am 15.05.)	<b>9</b>
4.1	Reduktionssemantik . . . . .	9
4.2	Reduktionssemantik der Sprache WHILE . . . . .	9
4.3	Reduktionssemantik (schematisch) . . . . .	10
<b>5</b>	<b>Denotationelle Semantik</b> (Vorlesung 5 am 22.05.)	<b>11</b>
5.1	Allgemeines Prinzip . . . . .	11
5.2	Denotationelle Semantik . . . . .	11
<b>6</b>	<b>Axiomatische Semantik</b> (Vorlesung 6 am 05.06.)	<b>13</b>
6.1	Allgemeine Methode . . . . .	13
6.2	Konkretisierung am Beispiel von WHILE' . . . . .	13
<b>7</b>	<b>Mathematische Grundlagen</b>	
	<b>zur Konstruktion semantischer Bereiche</b> (Vorlesung 7 am 12.06.)	<b>15</b>
7.1	Den. Semantik . . . . .	15
7.2	Konsequenz . . . . .	16
7.3	Definition Semantischer Bereich, CPO . . . . .	16
7.4	Definitionen (monton, stetig, strikt, $[\rightarrow]$ ) . . . . .	16
7.5	Satz: minimaler Fixpunkt (Tarski 1955) . . . . .	17
7.6	Graphische Illustration von cpo's . . . . .	17
<b>8</b>	<b>Konstruktion Semantischer Bereiche (cpo's)</b> (Vorlesung 8 am 19.06.)	<b>19</b>
8.1	Abzählbare Mengen . . . . .	19
8.2	Kartesisches Produkt . . . . .	20
8.3	Summen . . . . .	20
8.4	endliche Folgen . . . . .	20

8.5	unendliche Folgen . . . . .	21
8.6	Funktionsraum . . . . .	21
8.7	Ausblick . . . . .	21
<b>9</b>	<b>Der getypte <math>\lambda</math>-Kalkül als Metasprache</b> (Vorlesung 9 am 26.06.)	<b>22</b>
9.1	Definition: Menge $A_\lambda$ der getypten $\lambda$ -Ausdrücke . . . . .	22
9.2	Konventionen (Verbesserung der Lesbarkeit) . . . . .	22
9.3	Formale Semantik der Metasprache: . . . . .	23
9.4	$\lambda$ -Kalkül (Semantische Reduktionsregeln) . . . . .	23
<b>10</b>	<b>Die Metasprache <math>\lambda</math>-Kalkül</b> (Vorlesung 10 am 03.07.)	<b>24</b>
10.1	Syntaktischer Zucker in der Metasprache . . . . .	24
10.1.1	Kombinatoren (Namen für Ausdrücke) . . . . .	24
10.1.2	Operationen . . . . .	24
10.1.3	Bedingte Ausdrücke . . . . .	24
10.1.4	Rekursionen . . . . .	25
10.2	Syntaktische Bereiche von WHILE flache cpo's . . . . .	25
10.2.1	Sematische Bereiche . . . . .	25
10.2.2	Semantikfunktionen . . . . .	26
<b>11</b>	<b>Fortsetzungssemantik</b> (Vorlesung 11 am 10.07.)	<b>27</b>
11.1	Problem . . . . .	27
11.2	Continuations (Fortsetzung) . . . . .	27
11.3	Fortsetzungssemantik von WHILE . . . . .	27
11.4	Beispiel . . . . .	28

# 1 Einführung (Vorlesung 1 am 17.04.)

## 1.1 Was sind Programmiersprachen?

**Definition 1.1** (Programmiersprachen).

*Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.*

Memo technischer Begriff -> z.B. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

**Syntax** einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)

- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

**Semantik** (diese Vorlesung)

- Bedeutung
- Interpretation

Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

**Fokus:** Formalisierung

**Pragmatik** Fragen nach dem Gebrauch und Zweck (Useability).

*Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)*

## 1.2 Mehrdeutigkeit in natürlichen Sprachen

**Synonyme** *Schloss, Schimmel, ...*

Auflösung durch Kontext (meist leicht und unproblematisch)

**Satzebene** *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*

Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

## 1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

### Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

### Entwicklung der Formalisierungsansätze

**operationale Semantik** (*Landin 1964*): Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnlich wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

**denotationelle Semantik** (*McCarthy 1962*): Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

**axiomatische Semantik** (Hoare 1969): Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel:  $\{Q\}P\{R\}$

## 1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

### 1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)

Elementare Einheiten

```

1 // ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
2 Z ::= 0 | 1 | ... | MAX | -1 | -2 | ... | MIN
3 // Wahrheitswerte BOOL
4 W ::= TRUE | FALSE
5 // Konstanten KON
6 K ::= Z | W
7 // Bezeichner bzw. Variablen mit Indizes
8 I ::= a | b | ... | z | a1 | a2 | ... | zi
9 // Operatoren
10 OP ::= + | - | * | / | mod
11 // boolesche Operatoren
12 BOP ::= < | > | = | !> | !< | !=

```

Zusätzliche Einheiten (induktiv)

```

1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | READ, für T1, T2 in TERM
3 // boolesche Terme BT
4 B ::= W | T1 BOP T2 | READ | not B
5 // Befehle (Zustandstransformation) COM
6 C ::= skip | I := T | C1; C | if B then C1 else C2 | while B do C |
   output T | output B

```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!

Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

- Typkonflikte
- Fehlerbehandlung
- Rekursion

### WHILE ist mehrdeutig?

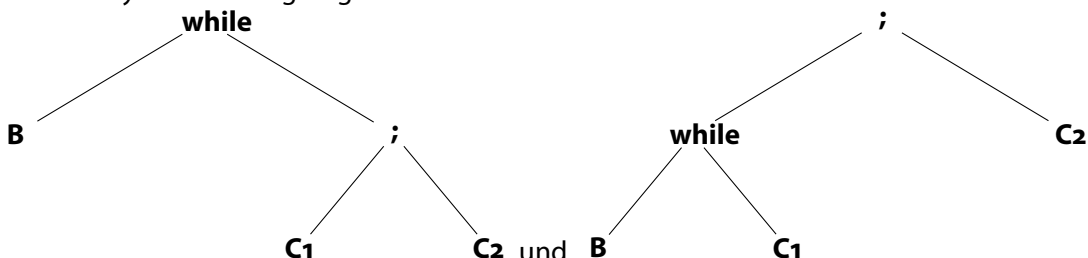
Ja, das zeigt folgendes Beispiel:

```

1 while B do C1; C2

```

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

---

```
1 while B do
2   C1;
3   C2
```

---

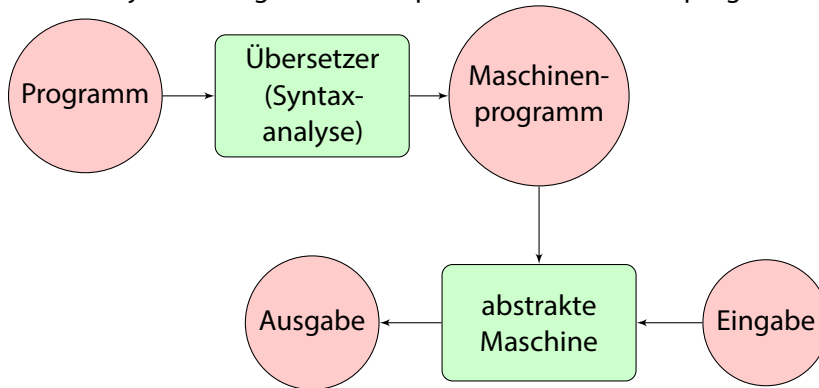
## 2 Operationelle Semantik (Vorlesung 2 am 24.04.)

### 2.1 Operationelle Semantik am Beispiel der Terme

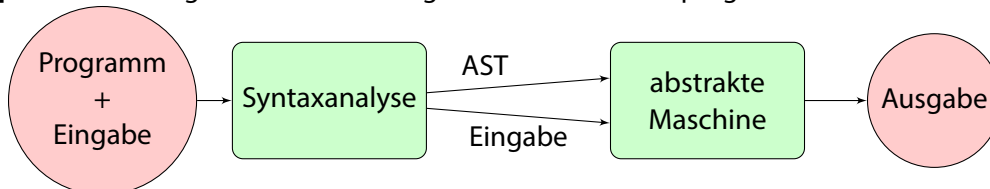
Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen! *(Inhalt ist nicht im Lehrbuch!)*

Grundsätzlich gibt es zwei Methoden:

**Übersetzer** Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



**Interpreter** Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



**AST:** abstract syntax tree

### 2.2 Terme der Sprache WHILE

---

```
1 // Terme TERM
2 T ::= Z | I | T1 OP T2 | READ, für T1, T2 in TERM
```

---

#### Beispiel: AST

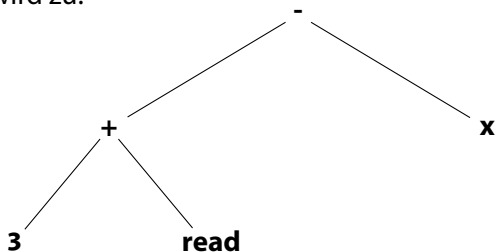
Der Ausdruck

---

```
1 3 + read - x
```

---

wird zu:

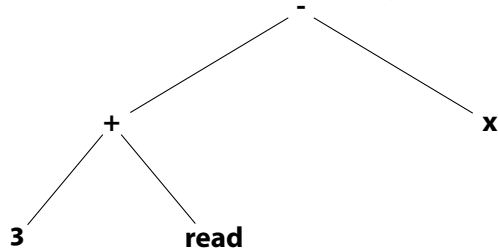


### 2.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.

**Annahme:** wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

**Übersetzer Idee:** depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

---

```

1 PUSH 3
2 READ
3 ADD
4 LOAD x
5 SUB
  
```

---

Zustandsveränderungen:

aktueller Zustand (siehe Bild Architektur!):

$\langle \epsilon   S   8.5 \dots \rangle$	$\xrightarrow{\text{PUSH } 3}$	$\langle 3   S   8.5 \dots \rangle$
	$\xrightarrow{\text{READ}}$	$\langle -8.3 - \epsilon   S   5 \dots \rangle$
	$\xrightarrow{\text{ADD}}$	$\langle 3 + (-8) \cdot \epsilon   S   5 \dots \rangle$
	$\xrightarrow{\text{LOAD } x}$	$\langle 2 - 5 \cdot \epsilon   S   \dots 5 \rangle$
	$\xrightarrow{\text{SUB}}$	$\langle -7 \cdot \epsilon   S   5 \dots \rangle$

Semantik eines Terms  $T$  zu geg. Speicher  $S$  und Eingabe  $E$  ist die Spitze des Wertekellers (STACK) nach Ausführung von  $\text{trans } T$  auf  $\langle \epsilon | S | E \rangle$ , falls diese Ausführung fehlerfrei läuft, sonst Fehler!

**Interpreter** (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
  - \* Wertekeller  $W$  ( $\in \text{ZAHL}^*$ )
  - \* Speicher  $S$  ( $S \in [ID \rightarrow \text{ZAHL}]$ )
  - \* Kontrollkeller  $K$  ( $\in (\text{AST} \cup \text{OP})^*$ )
  - \* Eingabe  $E$  ( $\in \text{ZAHL}^*$ )

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum  $Z$  durch Angabe von:

- (i) einem Anfangszustand  $Z_{T,S,E}$  für jeden Term  $T$ , Speicher  $S$  und Eingabe  $E$ .
- (ii) eine Zustandsüberföhrungsfunktion  $\Delta : Z \rightarrow Z$  (partiell)
- (iii) Erklärung der Semantik über Iteration von  $\Delta$

für Terme aus WHILE:

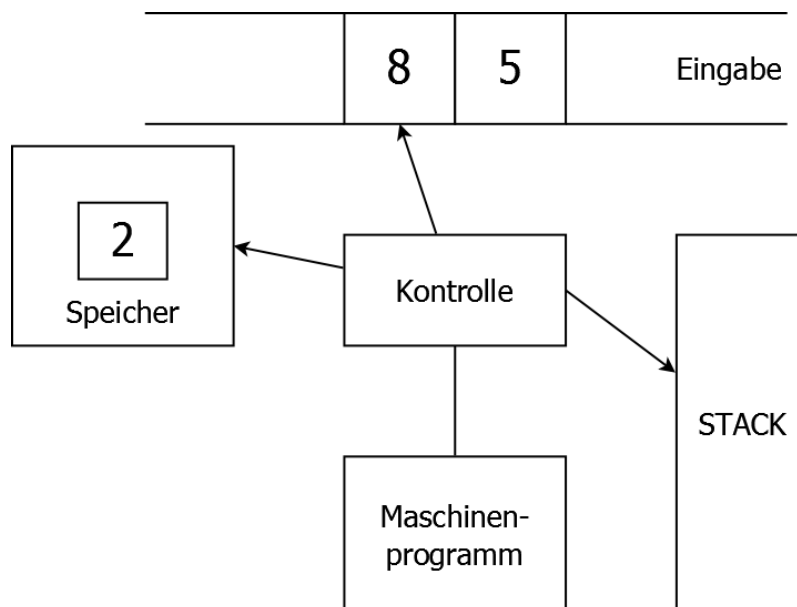
- (i)  $Z_{T_0, S_0, E_0} := \langle \epsilon | S_0 | T_0 \cdot \epsilon | E_0 \rangle$
- (ii)  $\Delta$  per Induktion über die Struktur der Kontrollkellerspitze



$$\begin{aligned}
\Delta \langle W|S|n.K|E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in Z\text{AHL}, W, S, E \text{ wie oben.} \\
\Delta \langle W|S|x.K|E \rangle &= \langle s(x).W|S|K|E \rangle \text{ für alle } x \in ID \\
\Delta \langle W|S|\text{READ}.K|n.E \rangle &= \langle n.W|S|K|E \rangle \text{ für alle } n \in Z\text{AHL} \\
\Delta \langle W|S|T_1OPT_2.K|E \rangle &= \langle W|S|T_1.T_2.OP.K|E \rangle \\
\Delta \langle n_2.n_1.W|S|OP.K|E \rangle &= \langle n_1OPn_2.W|S|K|E \rangle \text{ falls } n_1OPn_2 \text{ definiert ist.}
\end{aligned}$$

- (iii) Die Semantik eines (beliebigen) Terms  $T$  im Bezug auf einem Speicher  $S$  und eine Eingabe  $E$  ist  $n \in Z\text{AHL}$ , wenn  $\Delta^k Z_{T,S,E} = \langle n.\epsilon|S|\epsilon|E' \rangle$  für beliebige  $E' \in Z\text{AHL}^*$ , undefiniert sonst!

### Architektur der abst. Maschine



### Befehlssatz

---

```

1 // Stack und Speicher Operationen
2 READ      // lese Zahl v. Eingabe, PUSH Zahl, rücke Zeiger um eins weiter
3 LOAD x    // PUSH Inhalt aus Speicher mit symbol. Adr. x
4 PUSH n    // für jede Zahl aus N lege n auf Stack
5 (STORE x) // belegt Speicher mit der symbolischer Adresse x
6 (GOTO n)  // bedingter Sprung
7
8 // arithmetische Operationen
9 ADD
10 MULT
11 SUB

```

---

### 3 Operationelle Semantik (Vorlesung 3 am 08.05.)

(Mitschrift von HvB, weil  
Autor im Urlaub :)  
Layout nachträglich  
modifiziert von TH.)

#### 3.1 Allgemeines

Methodik einer Formalisierung (Interpreter) der Semantik einer Programmiersprache  $P$

Zur operationellen Semantik gehören insbesondere 3 Angaben:

1. Definition des Zustandsraums einer abstrakten Maschine - möglichst einfach:  $Z$
2. Definition einer (partiellen) Zustandsüberföhrungsfunktion  $\Delta : Z \rightarrow Z$
3. Definition eines Anfangszustands  $Z_{P,E}$  zu jedem Programm  $P$  und Eingabe  $E$

Aus 1.-3. ergibt sich die operationelle Semantik  $\mathcal{O}$  von  $\mathcal{P}$  wie folgt:

$\mathcal{O} : P \rightarrow [E \rightarrow A \cup \{\text{Fehler}\}]$

$$\mathcal{O}(P)(E) = \begin{cases} A, & \text{falls } \exists k \in \mathbb{N} \text{ mit } \Delta^k(Z_{P,E}) = \Delta^{k+1}(Z_{P,E}) \text{ und} \\ & A \text{ die Ausgabekomponente von } Z \text{ ist} \\ \text{Fehler,} & \text{falls es ein } k \in \mathbb{N} \text{ gibt mit } \Delta^k(Z_{P,E}) \text{ nicht definiert} \\ \text{undefiniert,} & \text{sonst.} \end{cases}$$

#### 3.2 Operationelle Semantik von WHILE

1. **Der Zustandsraum  $Z$  ist das kartesische Produkt  $\mathcal{W} \times \mathcal{S} \times \mathcal{K} \times \mathcal{E} \times \mathcal{A}$  mit:**

**Wertekeller**  $W \in \mathcal{W}$  ist eine Folge von Konstanten, d.h.

$\mathcal{W} = \text{KON}^*$

**Speicher**  $S \in \mathcal{S}$  ist eine Funktion von Bezeichnern nach  $\text{ZAHL} \cup \{\text{frei}\}$ , d.h.

$\mathcal{S} = [\text{ID} \rightarrow \text{ZAHL}]$

**Kontrollkeller**  $K \in \mathcal{K}$  ist ein Folge von ASTs bzw. Kontrollsymbolen, d.h.

$\mathcal{K} = (\text{TERM} \cup \text{TCOM} \cup \text{OP} \cup \text{BOP} \cup \{\text{if}, \text{while}, \text{assign}\})^*$

**Ein- und Ausgabe**  $E \in \mathcal{E}$  bzw.  $A \in \mathcal{A}$  ist jeweils eine Folge von Konstanten, d.h.

$\mathcal{E} = \text{KON}^*$  bzw.  $\mathcal{A} = \text{KON}^*$

2. **Induktion über den Aufbau der Kontrollkellerspitze**

##### a. TERM

$$T := Z | I | T_1 \text{ OP } T_2 | \text{read}$$

$$\Delta \langle W | S | n.K | E | A \rangle = \langle n.W | S | K | E | A \rangle \quad \text{für alle } n \in$$

$$\Delta \langle W | S | x.K | E | A \rangle = \langle S(x).W | S | K | E | A \rangle \quad \text{für alle } x \in \text{ID mit } S(x) \neq \text{frei}$$

$$\Delta \langle W | S | \text{read}.K | n.E | A \rangle = \langle n.W | S | K | E | A \rangle \quad \text{für alle } n \in \text{ZAHL}$$

$$\Delta \langle W | S | T_1 \text{ OP } T_2.K | E | A \rangle = \langle W | S | T_1.T_2.\text{OP}.K | E | A \rangle$$

$$\Delta \langle n_2.n_1.W | S | \text{OP}.K | E | A \rangle = \langle (n_1 \text{ OP } n_2).W | S | K | E | A \rangle, \quad \text{falls } n_1 \text{ OP } n_2$$

nicht aus dem darstellbaren Zahlenbereich herausführt

##### b. Boolsche Terme (ähnlich)

**c. COM**

$$C := \underline{skip} | I := T | C_1; C_2 | \underline{if} B \underline{then} C_1 \underline{else} C_2 | \\ \underline{while} B \underline{do} C | \underline{output} T | \underline{output} B$$

$$\Delta \langle W | S | \underline{skip}.K | E | A \rangle = \langle W | S | K | E | A \rangle$$

$$\Delta \langle W | S | I := T.K | E | A \rangle = \langle W | S | T.\underline{assign}.I.K | E | A \rangle$$

$$\Delta \langle W | S | I := T.K | E | A \rangle = \langle W | S | T.\underline{assign}.I.K | E | A \rangle$$

$$\Delta \langle n.W | S | \underline{assign}.I.K | E | A \rangle = \langle W | S | [n/I] | K | E | A \rangle,$$

$$\text{wobei } n \in \mathbf{ZHL}, S[n/I](x) = \begin{cases} n, & \text{falls } I = x \\ S(x) & \text{sonst} \end{cases}$$

$$\Delta \langle W | S | C_1; C_2.K | E | A \rangle = \langle W | S | C_1.C_2.K | E | A \rangle$$

$$\Delta \langle W | S | \underline{if} B \underline{then} C_1 \underline{else} C_2.K | E | A \rangle = \langle W | S | B.\underline{if}.C_1.C_2.K | E | A \rangle$$

$$\Delta \langle \underline{true}.W | S | \underline{if}.C_1.C_2.K | E | A \rangle = \langle W | S | C_1.K | E | A \rangle$$

$$\Delta \langle \underline{false}.W | S | \underline{if}.C_1.C_2.K | E | A \rangle = \langle W | S | C_2.K | E | A \rangle$$

$$\Delta \langle W | S | \underline{while} B \underline{do} C.K | E | A \rangle = \langle W | S | B.\underline{while}.B.C.K | E | A \rangle$$

$$\Delta \langle \underline{true}.W | S | \underline{while}.B.C.K | E | A \rangle = \langle W | S | C.B.\underline{while}.B.C.K | E | A \rangle$$

$$\Delta \langle \underline{false}.W | S | \underline{while}.B.C.K | E | A \rangle = \langle W | S | K | E | A \rangle$$

$$\Delta \langle W | S | \underline{output} T.K | E | A \rangle = \langle W | S | T.\underline{output}.K | E | A \rangle$$

$$\Delta \langle n.W | S | \underline{output}.K | E | A \rangle = \langle W | S | K | E | n.A \rangle$$

## 4 Reduktionssemantik (Vorlesung 4 am 15.05.)

### 4.1 Reduktionssemantik

Ausprägungen der operationellen Semantik zu einfacheren Argumentation (Beweisführung) über Programmeigenschaften.

**Idee:** Reduktion von Ausdrücken, Termen, Programmen und Anweisungen (usw.) auf einfachere aber semantisch äquivalenten Termen (usw.).

**Beispiel:** Einfache arithmetische Ausdrücke über den natürlichen Zahlen und  $+$ ,  $*$ .

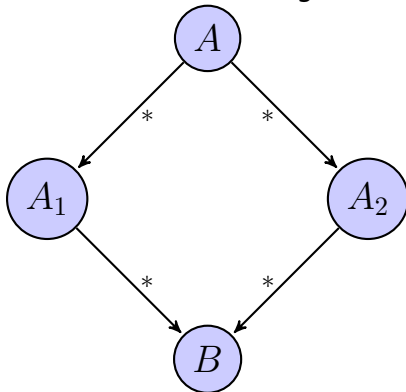
**Einzelschrittreduktion:**  $(4 + 2) * (7 - 5) \Rightarrow 6 * (7 - 5) \Rightarrow 6 * 2 \Rightarrow 12$

**allgemeine Form:**  $A_1 \underline{OP} A_2 \Rightarrow A'_1 \underline{OP} A'_2$  falls  $A_1 \Rightarrow A'_1$  und  $A_2 \Rightarrow A'_2$

**Axiom:**  $n_1 \underline{OP} n_2 \Rightarrow n_1 \underline{OP} n_2$

**Alternative Schreibweise(Winskel):**  $\frac{A_1 \Rightarrow A'_1, A_2 \Rightarrow A'_2}{A_1 \underline{OP} A_2 \Rightarrow A'_1 \underline{OP} A'_2}$

**Konfluenz:** (Church-Rosser Eigenschaft, Diamant)



Church-Rosser gilt nicht bei Nebenwirkungen!

anderes Beispiel:  $\lambda$ -Kalkül.  $(\lambda x.M)A \xrightarrow{\beta} M$  falls...

### 4.2 Reduktionssemantik der Sprache WHILE

Zustandsraum:  $\mathcal{Z} = \rho \times \mathcal{E} \times \mathcal{A}$  Speicher, Ein- und Ausgabe.  $\rho = [ID \rightarrow \text{ZAHL} \cup \{\text{frei}\}]$ ,  $\mathcal{E} = \text{KON}^*$ ,  $\mathcal{A} = \text{KON}^*$

$\Rightarrow$  Reduktionsrelation über  $(\text{TERM} \cup \text{BT} \cup \text{COM}) \times \mathcal{Z}$

Induktiv über den Aufbau der Syntax:

1. **Terme:** Keine Reduktionsregel für  $(n, z)$ , d.h. Normalform für  $n \in \text{ZAHL}$

**a**  $(x, (s, e, a)) \Rightarrow (s(x), (s, e, a))$ , falls  $s(x) \neq \text{frei}$  für  $x \in \text{ID}$ ,  $(s, e, a) \in \mathcal{Z}$

**b**  $\text{read} \Rightarrow (n, (s, e, a))$ , falls  $n \in \text{ZAHL}$ .

**c**  $(T_1 \underline{OP} T_2, z) \Rightarrow (n \underline{OP} T_2, z')$ , falls  $(T_1, z) \xRightarrow{*} (n, z')$

**d**  $(n \underline{OP} T_1, z) \Rightarrow (n \underline{OP} m, z')$ , falls  $(T, z) \xRightarrow{*} (m, z')$

**e**  $(n \underline{OP} n, z) \Rightarrow (n \underline{OP} m, z')$ , falls  $n \underline{OP} m \in \text{ZAHL}$ .

2. **BT** analog.

3. **COM:** keine Reduktionsregel für  $\text{skip}$  (Normalform)

**a**  $(I := T, (s, e, a)) \Rightarrow (\text{skip}, (s[n/I], e', a))$ , falls  $(T, (s, e, a)) \xRightarrow{*} (n, (s, e', a))$

**b**  $\text{output } T, (s, e, a) \Rightarrow (\text{skip}, (s, e', a.n))$ , falls  $(T, (s, e, a)) \xRightarrow{*} (n, (s, e', a))$

**c**  $\text{output } B, (s, e, a) \Rightarrow (\text{skip}, (s, e', a.b))$ , falls  $(B, (s, e, a)) \xRightarrow{*} (b, (s, e', a))$

**d**  $(C_1; C_2, z) \Rightarrow (C_2, z)$ , falls  $(C_1, z) \xRightarrow{*} (\text{skip}, z')$

**e**  $(\text{if } B \text{ then } C_1 \text{ else } C_2, z) \Rightarrow (C_1, z')$ , falls  $(B, z) \xRightarrow{*} (\text{true}, z')$

**f**  $(\text{if } B \text{ then } C_1 \text{ else } C_2, z) \Rightarrow (C_2, z')$ , falls  $(B, z) \xRightarrow{*} (\text{false}, z')$

**g**  $(\text{while } B \text{ do } C, z) \Rightarrow (C; \text{while } B \text{ do } C, z')$ , falls  $(B, z) \xRightarrow{*} (\text{true}, z')$

$$\mathbf{h} \quad (\underline{while} \ B \ \underline{do} \ C, z) \Rightarrow (\underline{skip}, z'), \text{ falls } (B, z) \xRightarrow{*} (\underline{false}, z')$$

### 4.3 Reduktionssemantik (schematisch)

$$\underline{eval}(P)(E) = \begin{cases} A, & \text{falls } (P, (S_0, E, \mathcal{E})) \xRightarrow{*} (\underline{skip}, (S, E', A)) \text{ mit bel. } S \in \rho \text{ und } E', A \in \text{KON}^* \\ \underline{\text{Fehler}}, & \text{falls } (P, (S_0, E, \mathcal{E})) \xRightarrow{*} (C, (S, E', A)) \text{ mit bel. } C \in \text{COM} \text{ und } E', A \in \text{KON}^* \\ & \text{und } C \neq \underline{skip} \text{ und } (C, (S, E', A)) \text{ lässt sich nicht mehr mit } \Rightarrow \text{reduzieren} \\ \text{undefiniert,} & \text{sonst.} \end{cases}$$

**Satz:**  $\mathcal{O} = \underline{eval}$  (extensional)

Beweis über strukturelle Induktion.

## 5 Denotationelle Semantik (Vorlesung 5 am 22.05.)

### Grundprinzip:

Direkte (injektive) Zuordnung von syntaktischen Strukturen zu deren Semantik (als mathematisches Objekt (Konstrukt)).

### 5.1 Allgemeines Prinzip

1. Ordne jeder syntaktischen Kategorie  $K$  der Sprache einen semantischen Bereich  $B_K$  zu.
2. Schreibe (definiere) zu jeder syntaktischen Kategorie eine Semantikfunktion  $\mathcal{K}$

$$\mathcal{K} : K \rightarrow B_K$$

#### Notation:

- Das Argument einer Semantikfunktion  $\mathcal{K}$  schreiben wir in  $\llbracket$  und  $\rrbracket$
- Funktionen z.B.  $f : A \rightarrow (B \rightarrow C)$  werden  $f \ a \ b = a \dots b \dots$

(Klammern sparen!)

### 5.2 Denotationelle Semantik

- **Syntaktische Kategorien** von WHILE: TERM, BT, COM, PROG
- Hilfsbereiche:

$$\begin{aligned} \text{ZUSTAND} &= \text{SPEICHER} \times \text{EINGABE} \times \text{AUSGABE} & \text{mit } \text{SPEICHER} &= \text{ID} \rightarrow \text{ZAHL} \cup \{\underline{\text{frei}}\} \\ & & \text{EINGABE} &= \text{KON}^* \\ & & \text{AUSGABE} &= \text{KON}^* \end{aligned} \quad \begin{array}{l} \text{(Abh. vom Speicher +} \\ \text{Nebenwirkung)} \end{array}$$

1.  $\mathcal{T} : \text{TERM} \rightarrow \underline{\underline{B_T}}$   
 $B_T := \text{ZAHL}???$

#### Exkurs: Curry-Isomorphie

$$\begin{array}{ccc} & \text{curry} & \\ & \curvearrowright & \\ (A \times B \rightarrow C) & \cong & A \rightarrow (B \rightarrow C) \\ & \curvearrowleft & \\ & \text{uncurry} & \end{array}$$

**curry:**  $f \ a \ b = f(a, b)$

**uncurry:**  $g \ (a, b) = g \ a \ b$

$$B_T := \text{SPEICHER} \rightarrow \text{EINGABE} \rightarrow ((\text{ZAHL} \times \text{EINGABE}) \cup \{\underline{\text{Fehler}}\})$$

$$\mathcal{B} : \text{BT} \rightarrow \text{SPEICHER} \rightarrow \text{EINGABE} \rightarrow ((\text{BOOL} \times \text{EINGABE}) \cup \{\underline{\text{Fehler}}\})$$

$$\mathcal{C} : \text{COM} \rightarrow \underbrace{\text{ZUSTAND} \rightarrow (\text{ZUSTAND} \cup \{\underline{\text{Fehler}}\})}_{B_C}$$

## 2. Induktive Definition über dem Aufbau

$$\begin{aligned}
\mathcal{T}[\![n]\!] s e &= (n, e) \quad \text{für alle } n \in \text{ZAHL} \\
\mathcal{T}[\![x]\!] s e &= \begin{cases} (n, e), & \text{falls } s x = n \\ \text{Fehler}, & \text{falls } s x = \text{frei für alle } x \in \text{ID} \end{cases} \\
\mathcal{T}[\![\text{read}]\!] s e &= \begin{cases} (n, e'), & \text{falls } e = n.e' \\ \text{Fehler}, & \text{falls } e = \mathcal{E} \text{ oder } e = b.e' \text{ mit } b \in \text{B00L} \end{cases} \\
\mathcal{T}[\![T_1 \text{ OP } T_2]\!] s e &= \begin{cases} (n, e''), & \text{falls } \mathcal{T}[\![T_1]\!] s e = (n_1, e') \text{ und} \\ & \mathcal{T}[\![T_2]\!] s e' = (n_2, e'') \text{ und} \\ & n_1 \text{ OP } n_2 = n \in \text{ZAHL} \\ \text{Fehler}, & \text{sonst} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathcal{B}[\![w]\!] s e &= (w, e) \quad \text{für alle } w \in \text{B00L} \\
\mathcal{B}[\![\text{not } B]\!] s e &= \begin{cases} (b, e'), & \text{falls } \mathcal{B}[\![B]\!] s e = (w, e') \text{ und } b = \neg w \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{B}[\![T_1 \text{ BOP } T_2]\!] s e &= \begin{cases} (b, e''), & \text{falls } \mathcal{T}[\![T_1]\!] s e = (n_1, e') \text{ und} \\ & \mathcal{T}[\![T_2]\!] s e' = (n_2, e'') \text{ und} \\ & n_1 \text{ BOP } n_2 = b \in \text{B00L} \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{B}[\![\text{read}]\!] s e &= \begin{cases} (b, e'), & \text{falls } e = b.e' \text{ mit } b \in \text{B00L} \\ \text{Fehler}, & \text{falls } e = \mathcal{E} \text{ oder } e = n.e' \text{ mit } n \in \text{ZAHL} \end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathcal{C}[\![\text{skip}]\!] z &= z & \text{alt. Schreibweise: } \mathcal{C}[\![\text{skip}]\!] z &= \text{id} \\
\mathcal{C}[\![I := T]\!] (s, e, a) &= \begin{cases} (S[n/I], e', a), & \text{falls } \mathcal{T}[\![T]\!] s e = (n, e') \\ \text{Fehler}, & \text{sonst} \end{cases} & z &= (s, e, a) \\
\mathcal{C}[\![\text{output } T]\!] (s, e, a) &= \begin{cases} (s, e', a.n), & \text{falls } \mathcal{T}[\![T]\!] s e = (n, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{output } B]\!] (s, e, a) &= \begin{cases} (s, e', a.b), & \text{falls } \mathcal{B}[\![B]\!] s e = (b, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![C_1; C_2]\!] z &= \begin{cases} \mathcal{C}[\![C_2]\!] z', \mathcal{C}[\![C_1]\!] z = z' & \text{mit } z' \in \text{ZUSTAND} \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{if } B \text{ then } C_1 \text{ else } C_2]\!] (s, e, a) &= \begin{cases} \mathcal{C}[\![C_1]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{true}, e') \\ \mathcal{C}[\![C_2]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{false}, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{C}[\![\text{while } B \text{ do } C]\!] (s, e, a) &= \begin{cases} \mathcal{C}[\![C; \text{while } B \text{ do } C]\!] (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{true}, e') \\ (s, e', a), & \text{falls } \mathcal{B}[\![B]\!] s e = (\text{false}, e') \\ \text{Fehler}, & \text{sonst} \end{cases} \\
\mathcal{P}[\![P]\!] e &= \begin{cases} a, & \text{falls } \mathcal{C}[\![C]\!] (s_0, e, \epsilon) = (s, e', a) \\ \text{Fehler}, & \text{falls } \mathcal{C}[\![C]\!] (s_0, e, \epsilon) = \text{Fehler} \\ \text{undefiniert}, & \text{sonst} \end{cases}
\end{aligned}$$

## 6 Axiomatische Semantik (Vorlesung 6 am 05.06.)

Am Beispiel von WHILE' (read  $I$  anstelle von Termen der Form read).

**Grundlage:** C. A. R. Hoare: *An aximatic Basis for Computer Programming* (CACM 1969)

(Hoare-Kalkül zum Bew. von Prog. Eigenschaft.)

### 6.1 Allgemeine Methode

**Annahme:** AST ist gegeben.

Neben der abstrakten Syntax brauchen wir vier Angaben:

1. Eine Menge von Bedingungen (*logische Ausdrücke, Prädikate, assertions*), die über den Zustandsraum interpretierbar sind. Im Allgemeinen Prädikatenlogik 1. Stufe oder Aussagenlogik.

(formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)

(formales Setting von logischen Ausdrücken vorgeben, damit man damit arbeiten kann!)

Beispiel:  $x < y, input = \epsilon, \exists i. x = p.i, \dots$  natürliche Interpretation: Verbinde  $I \in ID$  mit  $S(I)$  sowie input mit Eingabe....

Sei  $S(x) = 0$ , dann ist die Bedingung  $y/x = 7$  falsch.

2. Definition zu jeder atomaren Anweisung  $C$  ein Axiom bzw. ein Axiomenschema der Form  $\{Q\}C\{R\}$  - Hoare Formel

zu lesen als: Wenn die Bedingung  $Q$  auf einem Zustand  $z$  gilt und die Ausführung von  $C$  auf  $z$  in einem Zustand  $z'$  terminiert, dann gilt  $R$  auf  $z'$ .

Beispiel: für skip des Axiomenschema:  $\{Q\} \text{skip} \{Q\}$ , zu lesen: für alle  $Q$  gilt es.

3. Für zusammengesetzte Anweisungen  $C$  mit unmittelbaren Komponenten  $C_1, \dots, C_r$  eine Schlussregel der Form  $\frac{F_1, \dots, F_i}{\{Q\}C\{R\}}$ , wobei die  $F_i$  Hoare-Formeln zu  $C_1, \dots, C_r$  oder andere Formeln sind.

4. *Logischer Klebstoff*: Allgemeine Schlussregeln in Verbindung mit Hoare-Formeln, mindestens (oft ausreichend) die Konsequenzregel:  $\frac{Q \Rightarrow S, T \Rightarrow R, \{S\}C\{T\}}{\{Q\}C\{R\}}$ , wobei  $\Rightarrow$  die logische Herleitung bezeichnet. Daraus folgt: Beweisbar  $\Rightarrow$  Gültig. Frage: gilt diese Herleitung auch andersrum ( $F$  gültig  $\Rightarrow$  beweisbar)? Nein. Im Allgemeinen landet man beim Haltemaschinenproblem und dem Gödelschen Unvollständigkeitssatz.

### 6.2 Konkretisierung am Beispiel von WHILE'

$T ::= Z \mid I \mid T_1 \text{ OP } T_2$

$B ::= \text{true} \mid \text{false} \mid \text{eof} \mid T_1 \text{ Relop } T_2$

$C ::= \text{skip} \mid \text{read } I \mid I := T \mid \text{output } T \mid C_1; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid \text{while } B \text{ do } C$

1. Aussagenlogik:

– Terme über  $Z, I, OP, \text{input}, \text{output}$ , Konstruktor- und Selektorfunktionen ( $\cdot, \text{hd}, \text{tl}$ )

– Bedingungen:  $T_1 \text{ Relop } T_2$   
 $B_1 \vee B_2, B_1 \wedge B_2, \neg B$



2. atomare Definitionen:

$$\{Q\} \text{skip}\{Q\}$$

$$\{Q[T/I]\} I := T\{Q\},$$

(zu lesen: Für jede  
Bedingung  $Q$  gilt die  
Hoare-Formel  
 $\{Q\} \text{skip}\{Q\}$ )

(A.1)

wobei  $[T/I]$  die einmalige textliche Substitution von  
 $T$  für jedes Vorkommen von  $I$  in  $Q$  bezeichnet.

(A.2)

$$\{Q[\text{hd } \underline{\text{input}}/I \mid \text{tl } \underline{\text{input}}/\underline{\text{input}}] \text{ read } I\} Q \approx I := \text{hd}(\underline{\text{input}}); \underline{\text{input}} := \text{tl } (\underline{\text{input}})$$

(A.3)

$$\{Q[\text{output}.T/\underline{\text{output}}]\} \underline{\text{output}} T\{Q\} \approx \underline{\text{output}} = \underline{\text{output}}.T$$

(A.4)

Beispiele:

$$\{\text{input} = (4, 5) \wedge 2 + x = 3\} I := 2 + x \{\text{input} = (4, 5) \wedge I = 3\}$$

$$\{\text{hd } \underline{\text{input}} = 7 \wedge \text{tl } \underline{\text{input}} = (3)\} \text{ read } x \{x = 7 \wedge \text{input} = (3)\}$$

$$\{\underline{\text{output}}.7 + y = (1, 2, 3, 4, 5)\} \underline{\text{output}}(7 + 9) \{\underline{\text{output}} = (1, 2, 3, 4, 5)\}$$

3.

$$\frac{\{Q\}C_1\{S\}, \{S\}C_2\{R\}}{\{Q\}C_1, C_2\{R\}}$$

(A.5)

$$\frac{\{Q \wedge B\}C_1\{R\}, \{Q \wedge \neg B\}C_2\{R\}}{\{Q\} \text{if } B \text{ then } C_1 \text{ else } C_2\{R\}}$$

(A.6)

$$\frac{\{Q \wedge B\}C\{Q\}}{\{Q\} \text{while } B \text{ do } C\{Q \wedge \neg B\}}$$

(A.7)

4. Schlussregeln:

## 7 Mathematische Grundlagen zur Konstruktion semantischer Bereiche (Vorlesung 7 am 12.06.)

### 7.1 Den. Semantik

**Den. Semantik:** Zuordnung von mathematischen Objekten zu syntaktischen Konstrukten.

**Problem:** Rekursion. (Wdh.: Wir erklären eine Funktion (Zustand  $\rightarrow$  Zustand  $\cup$  Fehler)...

$$C[\underline{while} B \underline{do} C](s, e, a) = \begin{cases} C[C; \underline{while} B \underline{do} C](s, e', a), & \text{falls } \mathcal{B}[\underline{B}](s, e) = (\underline{true}, e') \\ (s, e', a), & \text{falls } \mathcal{B}[\underline{B}](s, e) = (\underline{false}, e') \\ \underline{Fehler}, & \text{sonst} \end{cases}$$

**Frage:** Gibt es eine eindeutige Lösung solcher Rekursionsgleichungen?

**Antwort:** I.A. nein.

**Beispiele:** (von leicht zu schwer)

$$f(x) = f(x) + 1 \quad \text{Keine Lösung im Bereich der totalen Funktionen } \mathbb{N} \rightarrow \mathbb{N}.$$

Eine Lösung im Bereich der partiellen Funktionen  $\mathbb{N} \rightarrow \mathbb{N}$

$f(x)$  ist undefiniert für alle  $x \in \mathbb{N}$ .

(Beispiel 1)

$$f(x) = \begin{cases} 0, & \text{falls } f(x) = 0 \\ 1, & \text{falls } f(x) \neq 0 \end{cases} \quad \begin{array}{l} \text{Drei Lösungen über partiellen Funktionen } \mathbb{N} \rightarrow \mathbb{N} : \\ \text{a) } n \mapsto 0 \\ \text{b) } n \mapsto 1 \\ \text{c) } f(n) \text{ ist nicht definiert für alle } n \in \mathbb{N} \end{array}$$

(Beispiel 2)

$$f(x) = \begin{cases} y, & \text{falls } f(x) = 0 \\ f(f(x, y - 1), f(x - 1, y)), & \text{sonst } f(x) \neq 0 \end{cases} \quad \begin{array}{l} \text{Lösungen im Bereich der} \\ \text{partiellen Funktionen } \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} : \\ \text{a) } (x, y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ \text{undefiniert sonst} \end{cases} \\ \text{b) } (x, y) \mapsto y \\ \text{c) } (x, y) \mapsto \max(x, y) \\ \text{d) } (x, y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ k, & \text{sonst für } k \in \mathbb{N}_+ \text{ bel.} \end{cases} \end{array}$$

(Beispiel 3)

Sei  $g : \mathbb{N} \rightarrow \mathbb{N}$  eine idempotente Funktion mit  $g(x) \neq 0$ , d.h.

$g(gx) = g(x)$ , dann ist

$$(x, y) \mapsto \begin{cases} y, & \text{falls } x = 0 \\ g(x), & \text{sonst} \end{cases}$$

(eine Lösung zu Beispiel 3)

## 7.2 Konsequenz

Konstruiere semantische Bereiche so, dass zu jeder rekursiven Gleichung eine *eindeutige Lösung* zugeordnet werden kann.

**Idee:**

- Betrachte totale Funktion über Bereichen, in denen ein Element undefiniert ( $\perp$ ) hinzugefügt ist.
- Betrachte eine Approximationsrelation  $\sqsubseteq$  (weniger definiert als), um unendliche Objekte durch endliche zu approximieren.

**Beispiel**

Part. Fkt.  $\mathbb{N} \rightarrow \mathbb{N}$  mit  $f \sqsubseteq g$  gdw  $\text{Graph}(f) \leq \text{Graph}(g)$

doppelt  $x = 2 * x$  wird approximiert von

$$f_{\sqsubseteq}(x) \begin{cases} 2 * x, & \text{falls } x < \sqsubseteq \\ \text{undefiniert sonst, } \sqsubseteq \in \mathbb{N} \end{cases}$$

## 7.3 Definition Semantischer Bereich, CPO

Eine Struktur  $\underline{A} = (A, \sqsubseteq_A)$  heißt semantischer Bereich (cpo), wenn 1-3 gilt:

*cpo - complete partial order*

1.  $\sqsubseteq_A$  ist eine Halbordnung auf  $A$  (reflexiv, transitiv, antisymmetrisch)
2. Es gibt bezüglich  $\sqsubseteq_A$  ein minimales Element  $\perp_A$  in  $A$ , d.h.  $\perp_A \sqsubseteq a$ , für alle  $a \in A$
3. Zu jeder Kette  $K \leq A$  existiert eine kleinste obere Schranke  $\bigsqcup K$  in  $A$ , wobei  $K$  Kette ist, wenn zu je zwei  $k_1, k_2 \in K$  gilt:  $k_1 \sqsubseteq_A k_2$  oder  $k_2 \sqsubseteq_A k_1$ .

**Beispiel**

$(\mathbb{N}, \leq)$  ist cpo? Nein! 1.  $\checkmark$  2.  $0 = \perp_{\mathbb{N}}$   $\checkmark$  3.  $\{x | x \text{ ist gerade}\}$  ist Kette in  $\mathbb{N}$  hat in  $\mathbb{N}$  keine kleinste obere Schranke!  $\times$

## 7.4 Definitionen (monoton, stetig, strikt, $[\rightarrow]$ )

Seien  $A$  und  $B$  cpo's.

- Eine Funktion  $f : A \rightarrow B$  heißt **monoton**, wenn  $f(a_1) \sqsubseteq_B f(a_2)$  für alle  $a_1 \sqsubseteq_A a_2 \in A$
- Eine Funktion  $f : A \rightarrow B$  heißt **stetig**, wenn  $f(K)$  eine Kette in  $B$  ist und  $f(\bigsqcup K) = \bigsqcup f(K)$  für alle  $K \subseteq A$  mit  $K$  ist Kette in  $A$ .
- Eine Funktion  $f : A \rightarrow B$  heißt **strikt**, wenn  $f(\perp_A) = \perp_B$
- $[A \rightarrow B]$  bezeichnet den Raum aller stetigen Funktionen von  $A \rightarrow B$ .

**1. Lemma**  $([A \rightarrow B], \sqsubseteq)$  ist cpo mit punktwiser Ordnung, d.h.  $f \sqsubseteq g$ , wenn  $f(a) \sqsubseteq_B g(a)$  für alle  $a \in A$

**2. Lemma** Aus  $f$  ist stetig folgt  $f$  ist monoton.

**Beweis:** Sei

$f : A \rightarrow B$  stetig und  $a_1 \sqsubseteq_A a_2 \in A$

$$f(a_1) \sqsubseteq \bigsqcup \{f(a_1), f(a_2)\} = \bigsqcup f(\{a_1, a_2\}) = f(\bigsqcup \{a_1, a_2\}) = f(a_2) \checkmark$$

## 7.5 Satz: minimaler Fixpunkt (Tarski 1955)

Sei

$f : A \rightarrow A$  eine stetige Funktion

Es existiert ein minimaler Fixpunkt

$\underline{fix} f$  in  $A$  und es gilt

$$\underline{fix} f = \bigsqcup_{\sqsubseteq \in \mathbb{N}} f^{\sqsubseteq}(\perp)$$

**Nachtrag:** Sei  $D$  ein cpo und  $f : D \rightarrow D$  eine stetige Funktion. Es existiert in  $D$  der minimale Fixpunkt,  $\underline{fix} f$ , von  $f$  in  $D$ .

Es gilt  $\underline{fix} f = \bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\perp)$ .

Beweis:

1.)

$$\begin{aligned} f\left(\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\perp)\right) &= \bigsqcup_{\mu \in \mathbb{N}} f^{\mu+1}(\perp) \text{ wg. Stetigkeit} \\ &= \bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\perp) \end{aligned}$$

Also ist  $\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\perp)$  ein Fixpunkt von  $f$ .

Nebenbemerkung:  
 $\{f^{\mu}(\perp) | \mu \in \mathbb{N}\}$  ist  
Kette in  $D$ .

Begründung:  
 $\perp, f(\perp), f^2(\perp), \dots, \perp \sqsubseteq f(\perp), f(\perp) \sqsubseteq f^2(\perp), f^2(\perp) \sqsubseteq f^3(\perp), \dots$  Minimalität  
von  $\perp$

2.) (Minimalität) Sei  $p \in D$  mit  $f(p) = p$

$\perp \sqsubseteq p$  Minimalität von  $\perp$

$f(\perp) \sqsubseteq f(p)$  Lemma: Stetigkeit  $\Rightarrow$  Monotonie

$f(\perp) \sqsubseteq f(p)$  für alle  $\mu \in \mathbb{N}$

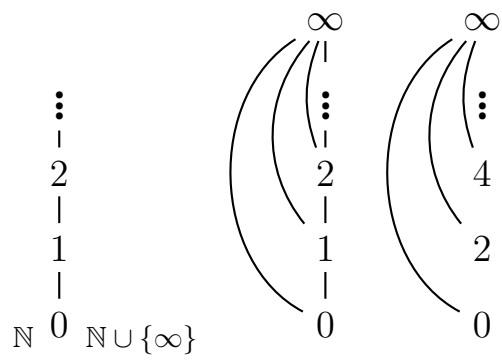
$$\bigsqcup_{\mu \in \mathbb{N}} f^{\mu}(\perp) \sqsubseteq p \quad \checkmark$$

□

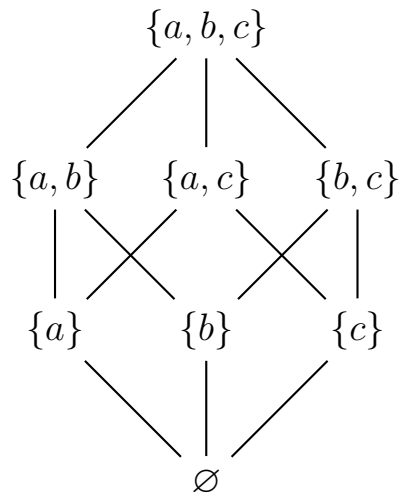
**Ausblick:** Rekursive Gleichungen können als Transformation von stetigen Funktionen betrachtet werden. *Fixpunkt dieser Transformation ist eindeutige Lösung.*

## 7.6 Graphische Illustration von cpo's

- Elemente des Trägers sind Knoten
- $\sqsubseteq$  wird durch aufwärts gerichtete Kanten dargestellt



Beispiel für cpo. Teilmengen von endlichen Mengen ( $\mathcal{P}\{a, b, c\}, \subseteq$ )



## 8 Konstruktion Semantischer Bereiche (cpo's) (Vorlesung 8 am 19.06.)

### 8.1 Abzählbare Mengen

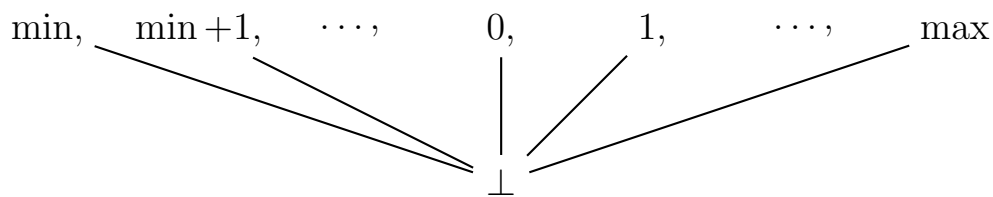
Sei  $M$  eine abzählbare Menge. Der cpo  $M_{\perp}$  entsteht aus  $M$  durch

$$M_{\perp} = (M \cup \{\perp\}, \sqsubseteq) \text{ mit } x \sqsubseteq y \text{ gdw. } x = \perp \text{ oder } x = y$$

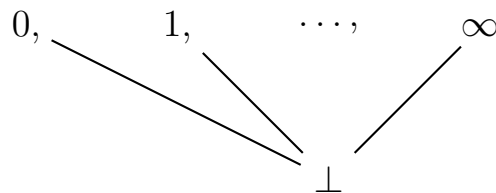
□

Beispiele:

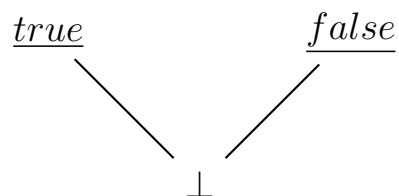
$$\text{ZAHL} = \{\min, \dots, \max\}_{\perp}$$



$$\mathbb{N}_{\perp} = 0, 1, 2, \dots$$



$$\text{BOOL} = \{\underline{true}, \underline{false}\}_{\perp}$$



$$\text{ID} = \{w \mid w \in \{a, b, \dots, z\}^*\}_{\perp}$$

## 8.2 Kartesische Produkt

Seien  $D_1, \dots, D_n$  cpo's.  $D = D_1 \times \dots \times D_n$

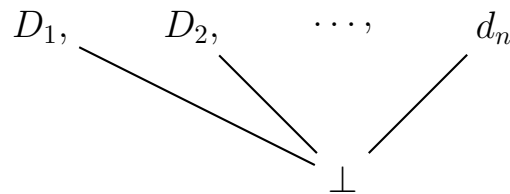
$D$  ist cpo mit komponentenweise Ordnung  $\sqsubseteq_D$ , d.h.

$$\langle d_1, \dots, d_n \rangle \sqsubseteq_D \langle d'_1, \dots, d'_n \rangle \text{ gdw. } d_i \sqsubseteq_{D_i} d'_i \text{ für alle } 1 \leq i \leq n$$

$$D := (D_1 \times D_n, \sqsubseteq_D) \text{ ist cpo.}$$

## 8.3 Summen

Seien:  $D_1, \dots, D_n$  cpo's



$D = D_1 + \dots + D_n$  ist wie folgt erklärt.

$D = (\{(d, i) | d \in D_i, 1 \leq i \leq n\} \cup \{\perp_D\}, \sqsubseteq_D)$  mit  $(d, i) \sqsubseteq_D (e, j)$  gdw.  $i = j$  und  $\perp_D \sqsubseteq (d, i)$  für alle ...

## 8.4 endliche Folgen

Sei  $\underline{D} = (D, \sqsubseteq_D)$  ein cpo.

$\underline{D}^* = (\{\langle d_i | d_i \in D, 1 \leq i \leq n \rangle | n \in \mathbb{N}\} \cup \{\perp_{D^*}\}, \sqsubseteq_{D^*})$  mit

Vorüberlegung

$$\langle d_i | 1 \leq i \leq n \rangle \sqsubseteq_D \langle d'_i | 1 \leq i \leq m \rangle$$

gdw.  $n \leq m$  und  $d_i \sqsubseteq_D d'_i$  für alle  $1 \leq i \leq n$

funktioniert nicht, weil

Sei  $1 \in D$ ,

$K := \{\langle \rangle, \langle 1 \rangle, \langle 1, 1 \rangle, \langle 1, 1, 1 \rangle, \dots\}$  Kette in  $D^*$

$\bigsqcup K = \langle 1, 1, 1, \dots \rangle$  unendliche Folgen in  $D^W$

Ende der Vorüberlegung

$$\langle d_i | 1 \leq i \leq n \rangle \sqsubseteq_{D^*} \langle d'_i | 1 \leq i \leq m \rangle$$

gdw.  $n = m$  und  $d_i \sqsubseteq_D d'_i$  für alle  $1 \leq i \leq n$

$\perp_{D^*} \sqsubseteq_{D^*}$  für jede Folge  $\in D^*$

$$\text{d.h. } D^* = \bigcup_{i \in \mathbb{N}} D^i$$

## 8.5 unendliche Folgen

$D^w := (\{ \langle d_i | i \in \mathbb{N} \rangle \}, \sqsubseteq_{D^w})$  mit  $\langle d_i | i \in \mathbb{N} \rangle \sqsubseteq_{D^w} \langle d'_i | i \in \mathbb{N} \rangle$  gdw.  $d_i \sqsubseteq_D d'_i$  für alle  $i \in \mathbb{N}$   
 $\perp_{D^w} = \langle \perp_D | i \in \mathbb{N} \rangle$

## 8.6 Funktionenraum

Seien  $D_1$  und  $D_2$  cpo's

$D = [D_1 \rightarrow D_2] := (\{ f : D_1 \rightarrow D_2 \mid f \text{ ist stetig} \}, \sqsubseteq_D)$  mit punktweiser Ordnung, d.h.  
 $f \sqsubseteq_D g$  gdw.  $f(d) \sqsubseteq_{D_2} g(d)$  für alle  $d \in D_1$ .

$D$  ist cpo!

## 8.7 Ausblick

Konstruktionen rekursiver Domains ist möglich. Wenn  $D$  eine cpo-Variable, dann löst sich eine Gleichung der Form  $D = \tau[D, B_1, \dots, B_r]$  lösen.  $\tau$  über  $\times, +, *, ^\omega, \rightarrow$ , cpo's  $B_i$   
Kanonische Operationen, wie Projektionen, Injektionen, Test auf leere Liste, ..., sind stetig.



## 9 Der getypte $\lambda$ -Kalkül als Metasprache (Vorlesung 9 am 26.06.)

- Semantische Bereiche, cpo's
- Bezeichnung für Elemente aus Ausgangsbereichen, kartesische Produkte, Folgen, Summenbereichen ist die vertraute mathematische Notation. Für Elemente aus Funktionenbereichen (-domains) bietet sich der  $\lambda$ -Kalkül von Alonso Church (1936) an.  
Bisher:  $f : D_1 \rightarrow D_2$  wurde definiert durch  $d \rightarrow t$ , wobei in  $t$  sowohl der formale Parameter  $d$  als auch bekannte Elemente vorkommen können.  
Problematisch für Funktionen höheren Typs.

### 9.1 Definition: Menge $A_\lambda$ der getypten $\lambda$ -Ausdrücke

Sei  $\mathcal{X} = \{\mathcal{X}^D \mid D \in \mathcal{D}, \mathcal{D}\}$  Familie von cpo's, die Ausgangsbereiche enthält und abgeschlossen ist unter:  $x, *, ^\omega, +$  und  $\rightarrow$ .

1.

$x : D \in \mathcal{A}_\lambda$  für alle  $x \in \mathcal{X}^D$

(Atome, Variable)

$k : D \in \mathcal{A}_\lambda$  für alle  $k \in \mathcal{K}^D$ , wobei

$\mathcal{K} = \{\mathcal{K}^D \mid D \in \mathcal{D}\}$ ,

die diskrete Elemente, sowie Projektionsfunktion,

Listenoperationen, fix, curry, ...

typfrei: Sei  $\mathcal{X}$  als unendl. Menge

(i)  $x \in \mathcal{X}$  ist  $\lambda$ -Ausdruck (Atom)

(ii)  $(t_1 t_2)$  ist  $\lambda$ -Ausdruck (Applikation)

(iii)  $(\lambda x.t)$  ist  $\lambda$ -Ausdruck (Abstraktion)

2.

$\langle t_1, \dots, t_r \rangle : D_1 \times \dots \times D_r \in \mathcal{A}_\lambda$  für alle  $t_i : D_i \in \mathcal{A}_\lambda, 1 \leq i \leq r$  (Tupel)

3.

$(t_1 t_2) : D \in \mathcal{A}_\lambda$ , falls  $t_1 : D' \rightarrow D \in \mathcal{A}_\lambda$  und  $t_2 : D' \in \mathcal{A}_\lambda$

(Applikation)

4.a)

$(\lambda x.t) : D_1 \rightarrow D_2 \in \mathcal{A}_\lambda$ , für alle  $x \in \mathcal{X}^{D_1}$  und  $t : D_2 \in \mathcal{A}_\lambda$

(monadisch)

4.b)

$(\lambda(x_1, \dots, x_r).t) : D_1 \times \dots \times D_r \rightarrow D$ , für  $x_i \in \mathcal{X}^{D_i}, t : D \in \mathcal{A}_\lambda, 1 \leq i \leq r$

(polyadisch)

### 9.2 Konventionen (Verbesserung der Lesbarkeit)

- Applikation ist links assoziativ, d.h.  $f a_1 a_2$  steht für  $(\dots ((f a_1) a_2) \dots a_n)$
- Abstraktion erstreckt sich soweit nach rechts wie möglich, d.h.  $\lambda x.t_1 t_2 \dots t_r$  steht für  $(\lambda x.t_1 t_2 \dots t_r)$  und  $t_0(\lambda x.t_1 t_2 \dots t_r)t_{r+1}$  steht für  $(t_0(\lambda x.t_1 t_2 \dots t_r))t_{r+1}$
- Applikation mit rechtsassoziativer Bindung notiert durch  $;;$  d.h.  $t_1 \dots t_r; s_1 \dots s_n; u_1 \dots u_m$  steht für  $(t_1 \dots t_r)((s_1 \dots s_n)(u_1 \dots u_m))$
- Mehrfachabstraktionen der Form  $\lambda x_1.\lambda x_2 \dots \lambda x_r.t$  wird abgekürzt durch  $\lambda x_1 \dots x_r.t$
- Die bekannten zweistelligen arithm. u. log. Operatoren werden Infix notiert und binden schwächer als die Applikation, d.h.  $\lambda x.2 + fx$  steht für  $(\lambda x.(\underline{plus}2)(fx))$
- Verzicht auf Typenangabe, wenn aus Kontext ersichtlich!

## Beispiel

Für die Verwendung getypter  $\lambda$ -Ausdruck zur Definition von Elementen aus funktionalen cpo's.  
**Verbal** Gewicht  $f$ , welches angewendet auf eine Liste von Zahlen, eine zweistellige arithm. Operation  $g$  und eine Zahl  $z$  den Wert von  $g$  angewendet auf die dritte Komponente von  $L$  und  $z$  liefert.

$$f := \lambda L. g \ z. g(\pi_3 L) z \text{ mit } g : \mathbb{Z}_\perp \rightarrow \mathbb{Z}_\perp \rightarrow \mathbb{Z}_\perp, L : D^* \text{ oder } L : \mathbb{Z}_\perp^*, z : \mathbb{Z}_\perp$$

$$f : \mathbb{Z}_\perp^* \rightarrow [\mathbb{Z}_\perp \rightarrow \mathbb{Z}_\perp \rightarrow \mathbb{Z}_\perp] \rightarrow \mathbb{Z}_\perp \rightarrow \mathbb{Z}_{perp}$$

gegeben folgenden Argumente  $f < 4, 7, \underline{hd} < 3 >, 8, 2 > \underline{plus} 5 : \mathbb{Z}_\perp$   
 Rechnen durch Ersetzung von formalen Parametern durch akt. Argumente

$$\begin{aligned} f < 4, 7, \underline{hd} < 3 >, 8, 2 > \underline{plus} 5 &\rightarrow \underline{plus}(\pi_3 < \dots >) 5 \\ &\rightarrow \underline{plus}(\underline{hd} < 3 >) 5 \\ &\rightarrow \underline{plus} 3 5 \\ &\rightarrow 8 \end{aligned}$$

## 9.3 Formale Semantik der Metasprache:

$\llbracket \cdot \rrbracket : \mathcal{A}_\lambda \rightarrow [\mathcal{U} \rightarrow \mathcal{D}]$ , wobei  $\mathcal{U} : \mathcal{X} \rightarrow \mathcal{D}$  typhaltende Funktion.

$$\llbracket x \rrbracket \rho = \rho \ x$$

$$\llbracket k \rrbracket \rho = k \tag{1}$$

$$\llbracket < t_1, \dots, t_n > \rrbracket \rho = < \llbracket t_1 \rrbracket \rho, \dots, \llbracket t_n \rrbracket \rho > \tag{2}$$

$$\llbracket t_1 \ t_2 \rrbracket \rho = \llbracket t_1 \rrbracket \rho (\llbracket t_2 \rrbracket \rho) \tag{3}$$

$$\llbracket \lambda x. t \rrbracket \rho = d \rightarrow \llbracket t \rrbracket s[d/x] \tag{4a}$$

$$\begin{aligned} \llbracket \lambda(x_1, \dots, x_r). t \rrbracket \rho &= < d_1, \dots, d_r > \\ &\rightarrow \llbracket t \rrbracket \rho[d_1 \dots d_r / x_1 \dots x_i], x_i \neq x_j \text{ für alle } i \neq j \end{aligned} \tag{4b}$$

## 9.4 $\lambda$ -Kalkül (Semantische Reduktionsregeln)

$\text{Fr} : \mathcal{A}_\lambda \rightarrow \mathcal{X}$ ,  $\text{Geb} : \mathcal{A}_\lambda \rightarrow \mathcal{X}$ ,  $\text{Var} : \mathcal{A}_\lambda \rightarrow \mathcal{X}$

$\xrightarrow{\alpha}$  Variablenumbenennung  $\lambda x. t \xrightarrow{\alpha} \lambda y. \$^x_y t$ , wobei  $\$$  Substitutionsoperator

$\xrightarrow{\beta}$  Ersetzung formaler Parameter  $(\lambda x. t) a \xrightarrow{\beta} \$^x_a t$ , falls  $\text{Fr}(a) \text{Geb}(t) = \emptyset$

$\xrightarrow{\gamma}$  Wertreduktion,  $\underline{plus} \ 4 \ 3 \xrightarrow{\gamma} 7$

Extensionalität:  $\lambda x. (t \ x) \xrightarrow{\eta} t, x \notin \text{Fr}(t)$

## 10 Die Metasprache $\lambda$ -Kalkül (Vorlesung 10 am 03.07.)

### 10.1 Syntaktischer Zucker in der Metasprache

#### 10.1.1 Kombinatoren (Namen für Ausdrücke)

$\underline{id} = \lambda x.x$  Familie von Identitätsfunktion des Typs  $D \rightarrow D$  für alle  $D \in D$

$\underline{curry} : [(D_1 \times D_2) \rightarrow D_3] \rightarrow [D_1 \rightarrow D_2 \rightarrow D_3]$

$\underline{curry} = \lambda f x y. f \langle x, y \rangle$  Schreibe  $\underline{curry} f$  zur Transformation von  $f$

$\vdots$

**Konvention:** Schreibe äußere Abstraktionen nach links, z.B.:

$\underline{id} x = x$

$\underline{curry} f x y = f \langle x, y \rangle$

Randbemerkung:

$S = \lambda xyz.(xz)yz$

$K = \lambda xy.x$

$I = \lambda x.x$

und Applikation bilden  
berechenbare  
Funktionen

#### 10.1.2 Operationen

Im Falle der 2-Stelligkeit auch in Infix-Notation.

$\circ : ([D_2 \rightarrow D_3] \times [D_1 \rightarrow D_2]) \rightarrow [D_1 \rightarrow D_3]$

$(f \circ g) x = f(g x)$  alternativ:  $\circ = \lambda(f, g)x.f(g, x)$

$\bowtie$ -Operator (gelesen "vor"), der Fehler weiterreicht und implizit  $\underline{curry}$  anwendet.

**1. Fall**  $f : D_1 \rightarrow (D_2 + \{\underline{Fehler}\}_{\perp})$  und  $g : D_2 \rightarrow (D_3 + \{\underline{Fehler}\}_{\perp})$  oder  $g : D_2 \rightarrow D_3$

**2. Fall**  $f : D_1 \rightarrow (D_2 \times \dots \times D_n) + \{\underline{Fehler}\}_{\perp}$  und  $g : D_2 \rightarrow \dots \rightarrow [D_n \rightarrow (D_{n+1} + \{\underline{Fehler}\}_{\perp})]$  oder

$g : D_2 \rightarrow \dots \rightarrow D_n \rightarrow D_{n+1}$

zu 1.:  $(f \bowtie g)x = f x = \underline{Fehler}, \underline{Fehler}$

$f x = d, g h$

$\bowtie : [D_1 \rightarrow (D_2 + \{\underline{Fehler}\}_{\perp})] \times [D_2 \rightarrow (D_3 + \{\underline{Fehler}\}_{\perp})] \rightarrow [D_1 \rightarrow D_3 + \{\underline{Fehler}\}_{\perp}]$

analog für  $g : D_2 \rightarrow D_3$

zu 2.:  $(f \bowtie g) x = f x = \underline{Fehler}, \underline{Fehler};$

$f x = \langle d_2, \dots, d_n \rangle \rightarrow g d_2 \dots d_n$

#### 10.1.3 Bedingte Ausdrücke

$\underline{cond}$  ist nicht strikt!

$\underline{cond} : (D \times D) \rightarrow \text{BOOL} \rightarrow D$

$\underline{cond} \langle d_1, d_2 \rangle b = \begin{cases} b = \underline{true} & \rightarrow d_1, \\ b = \underline{false} & \rightarrow d_2, \perp \end{cases}$

### Integration im getypten $\lambda$ -Kalkül

Verwende  $D \rightarrow D \rightarrow D$  anstelle von  $\text{BOOL}$

kodiere  $\underline{true} = \lambda xy.x$   $\underline{false} = \lambda xy.y$   $\underline{cond} \langle d_1, d_2 \rangle b = b d_1 d_2$

## Musteranpassung in bedingten Ausdrücken

Wenn  $f$  aus einem Summebereich mit strukturell verschiedenen Gliedern besteht, werden in- und out- und Termfunktionen implizit angewendet, z.B.

$$\begin{aligned} f &: D_1 + (D_2 \times D_3) \rightarrow D \\ f\ x = x &= d \rightarrow \dots d \dots, \\ x &= \langle d_1, d_2 \rangle \rightarrow \dots d_1 \dots d_2 \dots \end{aligned}$$

Wenn  $f$  nicht substituiert wird auch:

$$\begin{aligned} f\ d &= \dots d \\ f\ \langle d_1, d_2 \rangle &= \dots d_1 \dots d_2 \dots \end{aligned}$$

### 10.1.4 Rekursionen

$f = A$  mit  $f \in \mathcal{X}^D$ ,  $A : D \in \mathcal{A}_\lambda$  definiert eindeutig ein Element in  $D$ .

wenn  $f$  in  $A$  frei vorkommt ist  $f = A$  eine rekursive Gleichung, deren eindeutige Lösung durch fix  $\tau$ , wobei  $\tau = \lambda f. A$  gegeben ist!

Ordne  $f = A$  die Transformation  $\tau = \lambda f. A$  zu.  $\tau : D \rightarrow D$ .

$$\begin{aligned} \underline{fac}\ n &= n = 0 \rightarrow 1, n * \underline{fac}(n-1); \underline{fac} : \mathbb{N}_\perp \rightarrow \mathbb{N}_\perp \\ \tau &= \lambda f n. n = 0 \rightarrow 1, n * f(n-1) \\ \tau : [\mathbb{N}_\perp \rightarrow \mathbb{N}_\perp] &\rightarrow [\mathbb{N}_\perp \rightarrow \mathbb{N}_\perp] \text{ mit } \underline{fix}\ \tau : \mathbb{N}_\perp \rightarrow \mathbb{N}_\perp \end{aligned}$$

Behauptung: fix  $\tau = !$

$$\begin{aligned} (\underline{fix}\ \tau) &= \bigsqcup_{\mu \in \mathbb{N}} \tau^\mu(\perp) \\ \tau^0(\perp) &= \lambda n. \perp \\ \tau^1(\perp)n &= n = 0 \rightarrow 1, \perp \\ \tau^2(\perp)n &= n = 0 \rightarrow 1, n = 1 \rightarrow 1, \perp \\ \tau^\mu(\perp)n &= n = 0 \rightarrow 1, n = 1 \rightarrow 1, \dots, n = \mu - 1 \rightarrow (\mu - 1), \perp \\ \tau^{\mu+1}(\perp)n &= \dots \end{aligned}$$

## 10.2 Syntaktische Bereiche von WHILE flache cpo's

### 10.2.1 Semantische Bereiche

$$\begin{aligned} \text{SPEICHER} &= \text{ID} \rightarrow (\text{ZAHL} + \{\underline{frei}\}_\perp) \\ \text{EINGABE} &= \text{KON}^*, \text{AUSGABE} = \text{KON}^* \end{aligned}$$

### 10.2.2 Semantikfunktionen

$$\mathcal{T} : \text{TERM} \rightarrow \text{ZUSTAND} \rightarrow ((\text{ZAHL} \times \text{ZUSTAND}) + \{\underline{Fehler}\}_{\perp})$$

$$\mathcal{B} : \text{TERM} \rightarrow \text{ZUSTAND} \rightarrow ((\text{BOOL} \times \text{ZUSTAND}) + \{\underline{Fehler}\}_{\perp})$$

$$\mathcal{C} : \text{COM} \rightarrow \text{ZUSTAND} \rightarrow (\text{ZUSTAND} + \{\underline{Fehler}\}_{\perp})$$

$$\mathcal{P} : \text{PROG} \rightarrow \text{EINGABE} \rightarrow (\text{EINGABE} + \{\underline{Fehler}\}_{\perp})$$

$$\mathcal{T}[\![n]\!]z = \langle n, z \rangle$$

für alle  $n \in \text{ZAHL}$

$$\mathcal{T}[\![x]\!] \langle s, e, a \rangle = s \ x = \underline{frei} \rightarrow \underline{Fehler}, \langle s \ x, \langle s, e, a \rangle \rangle$$

$$\mathcal{T}[\![read]\!] \langle s, e, a \rangle = \underline{null} \ e \rightarrow \underline{Fehler}, \langle \underline{hd} \ e, \langle s, \underline{tl} \ e, a \rangle \rangle$$

$$\mathcal{T}[\![T_1 + T_2]\!] = \mathcal{T}[\![T_1]\!] \bowtie \lambda n_1. \mathcal{T}[\![T_2]\!] \bowtie \lambda n_2 z. \langle n_1 + n_2, z \rangle$$

$$\mathcal{C}[\![C_1, C_2]\!] = \mathcal{C}[\![C_1]\!] \bowtie \mathcal{C}[\![C_2]\!]$$

## 11 Fortsetzungssemantik (Vorlesung 11 am 10.07.)

KLAUSUR am 17.7. 8-10  
Uhr, Raum 005, kein  
Material!

### 11.1 Problem

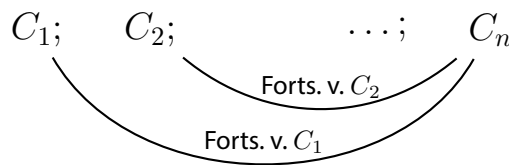
Sprunganweisungen der Form goto  $L$  oder andere irreguläre Programmfortsetzungen.

**Idee:**  $P = C_1; C_2; C_3; \dots; C_n$

Betrachte fehlerfreien Fall für  $C_1; C_2$ :

Bisher:  $\mathcal{C}[[C_1; C_2]]z = \mathcal{C}[[C_2]](\mathcal{C}[[C_1]])$

### 11.2 Continuations (Fortsetzung)



Die Fortsetzung von  $C_1$   
ist Zustandstransforma-  
tion definiert durch die  
Ausführung der Reste  
 $C_2; \dots; C_n$

(Wadsworth, Strachey 1974)

Jetzt hat  $\mathcal{C}[[C_i]]c_i z$  Zugriff auf die zug. Fortsetzung.

Dann kann  $\mathcal{C}[[\text{goto } L]]c z = C_L z$  kürzer  $\mathcal{C}[[\text{goto } L]]c = C_L$ , wobei  $C_L$  die Fortsetzung ist, die bei der Marke  $L$  beginnt.

Bei  $\mathcal{C}[[C_1; C_2]]c = \mathcal{C}[[C_1]](\mathcal{C}[[C_2]] c)$

### 11.3 Fortsetzungssemantik von WHILE

Syntaktische Bereiche und semantische Bereiche ZAHL, BOOL, KON, ZUSTAND, SPEICHER, EINGABE, AUSGABE, ... wie zuvor.

FORTSETZUNG = ZUSTAND  $\rightarrow$  (ZUSTAND + Fehler<sub>⊥</sub>) für COM

TERMFORT = ZAHL  $\rightarrow$  ZUSTAND  $\rightarrow$  (ZUSTAND + Fehler<sub>⊥</sub>) für TERM

= ZAHL  $\rightarrow$  FORTSETZUNG

BOOLFORT = BOOL  $\rightarrow$  FORTSETZUNG für BT

Schreibe die Typen für die Semantikfunktionen auf.

$\mathcal{C} : \text{COM} \rightarrow \text{FORTSETZUNG} \rightarrow (\text{ZUSTAND} + \text{Fehler}_{\perp})$

(kurz)  $\mathcal{C} : \text{COM} \rightarrow \text{FORTSETZUNG} \rightarrow \text{FORTSETZUNG}$

$\mathcal{T} : \text{TERM} \rightarrow \text{TERMFORT} \rightarrow \text{FORTSETZUNG}$

$\mathcal{B} : \text{BT} \rightarrow \text{BOOLFORT} \rightarrow \text{FORTSETZUNG}$

semantische Klauseln:

$$\mathcal{T}[\![n]\!]k = k\ n$$

$$\mathcal{T}[\![x]\!]k < s, e, a > = sx = \underline{frei} \rightarrow \underline{Fehler}, k(sx) < s, e, a >$$

$$\mathcal{T}[\![read]\!]k < s, e, a > = e \neq < > \rightarrow (\underline{isZahl}(hd\ e) < s, tl\ e, a >), \underline{Fehler}$$

$$\mathcal{T}[\![T_1 + T_2]\!]k = \mathcal{T}[\![T_1]\!]\lambda n_1. \mathcal{T}[\![T_2]\!]\lambda n_2. z.k(n_1 + n_2) | \text{Semantik über } \mathbb{Z}_\perp$$

$$= \mathcal{T}[\![T_1]\!]\lambda n_1. \mathcal{T}[\![T_2]\!]\lambda n_2. \min > (n_1 + n_2) \vee \max < (n_1 + n_2) \rightarrow \underline{Fehler}, k(n_1 + n_2)z$$

$$\mathcal{C}[\![skip]\!] = \underline{id} \quad \text{alternativ} \quad \mathcal{C}[\![skip]\!]c = c$$

$$\mathcal{C}[\![I := T]\!]c = \mathcal{T}[\![T]\!]\lambda n(s, e, a). c < s[n/I], e, a >$$

$$\mathcal{C}[\![output\ T]\!]c = \mathcal{T}[\![T]\!]\lambda n(s, e, a). c < s, e, a.n >$$

$$\mathcal{C}[\![C_1; C_2]\!]c = \mathcal{C}[\![C_1]\!] \circ \mathcal{C}[\![C_2]\!]$$

$$\mathcal{C}[\![if\ B\ then\ C_1\ else\ C_2]\!] = \mathcal{B}[\![B]\!](\underline{cond} < \mathcal{C}[\![C_1]\!], \mathcal{C}[\![C_2]\!] >)$$

$$\mathcal{C}[\![while\ B\ do\ C]\!] = \mathcal{B}[\![B]\!](\underline{cond} < \mathcal{C}[\![C]\!] \circ \mathcal{C}[\![while\ B\ do\ C]\!], \underline{id} >)$$

## 11.4 Beispiel

$$P = \underbrace{x = \underline{read}}_{C_1}; \underbrace{\underline{if\ } x > 0 \underline{then\ } \underbrace{\underline{output\ 1}}_{C_3} \underline{else\ } \underbrace{\underline{output\ (-1)}}_{C_4}}_{C_2}$$

$$\mathcal{P} : \text{COM} \rightarrow \text{EINGABE} \rightarrow (\text{AUSGABE} + \underline{Fehler}_\perp)$$

$$\mathcal{P}[\![P]\!]e = (\mathcal{C}[\![P]\!](\lambda z.z) \bowtie \pi_3) < s_0, e, \epsilon >$$

$$\mathcal{C}[\![P]\!]c_0\ z_0 = \mathcal{C}[\![C_1]\!](\underbrace{\mathcal{C}[\![C_2]\!]c_0}_{C_1})\ z_0 \quad \text{mit } z_0 = < s_0, < -24 >, \epsilon >$$

$$= \mathcal{T}[\![read]\!](\underbrace{(\lambda n(s, e, a). c_1 < s[n/x], e, a >)}_{k_1})\ z_0$$

$$= k_1(-24) < s_0, < >, \epsilon >$$

$$= c_1 < s_0[-24/x, \epsilon, \epsilon] >$$

⋮